Engineering a Generic Modular Mapping Framework

Philipp Helle and Wladimir Schamai
Airbus Group Innovations
Hamburg, Germany
Email: {philipp.helle,wladimir.schamai}@airbus.com

***Abstract*—This article presents a new framework for solving different kinds of data mapping problems, the Generic Modular Mapping Framework (GEMMA), and the engineering process that lead to its development. GEMMA is geared towards high flexibility for dealing with a large number of different challenges. To this end it has an open architecture that allows the inclusion of application-specific code and provides a generic rule-based mapping engine that allows users without programming knowledge to define their own mapping rules. The paper provides the thought processes that were involved in the engineering of the framework, detailed description of the concepts inherent in the framework and its current architecture. Additionally, the evaluation of the framework in two different application cases, simulation model composition and test bench setup, is described.**

Keywords–Mapping; Framework; Simulation Model Composition.

I. Introduction

This article is a revised and extended version of the article [1], which was originally presented at the The Seventh International Conference on Advances in System Simulation (SIMUL 2015).

Recently, several of our research challenges could be reduced to a common core question: How can we match data from one or more data sources to other data from the same and/or different data sources in a flexible and efficient manner? A search for an existing tool that satisfied our application requirements did not yield any results. This sparked the idea of a new common generic framework for data mapping. The goal in designing this framework was to create an extensible and user-configurable tool that would allow a user to define the rules for mapping data without the necessity for programming knowledge and that yet still has the possibility to include application-specific code to adapt to the needs of a concrete application.

Figure 1 shows an example, in which data points from different data sources have mapping relationships. A mapping problem can now be defined as the challenge to identify mappings between data points from (potentially) different data sources. This is what we want to automate.

The results of our efforts so far and a first evaluation based on our existing research challenges are presented in this paper.

This paper is structured as follows: Section II provides information regarding related work. Section III provides a detailed description of the framework, its core concepts and its architecture. Next, Section IV describes the application cases that have been used for developing and evaluating the framework so far. Finally, Section V concludes this article.

II. Related Work

The related work can be divided into two major categories: on the one hand, record linkage and data deduplication tools and frameworks, and on the other hand semantic matching frameworks for ontologies.

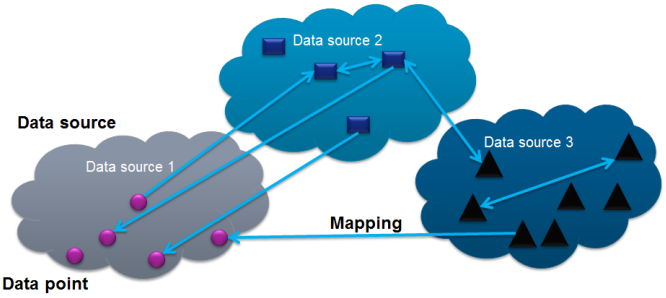

Figure 1. Data mapping

Record linkage as established by Dunn in his seminal paper [2] and formalised by Felligi and Sunter [3] deals with the challenge to identify data points that correspond with each other in large data sets. Typically, this involves databases of different origin and the question, which of the data on one side essentially are the same on the other side even if their name does not match precisely. The same approach is also called data deduplication [4] where the goal is to identify and remove redundancies in separate data sets. An overview of existing tools and frameworks can be found in [5]. The research work in that area focuses on efficient algorithms for approximate and fuzzy string matching since the size of the data sets involved often leads to an explosion of the run times. These tools [6] often include phonetic similarity metrics or analysis based on common typing errors, i.e., analysis based on the language of the input data. They concentrate on the matching of the string identifiers whereas our framework is more open and flexible in that regard and also includes the possibility to base the matching on available semantic meta-information. The goal in record linkage is always finding data points in different sets representing the same real-world object. Our framework was developed with the goal to match data from different sources that is related but not necessarily referencing the same object.

Semantic matching is a type of ontology matching technique that relies on semantic information encoded in ontologies to identify nodes that are semantically related [7]. They are mostly developed and used in the context of the semantic web [8], where the challenge is to import data from different heterogeneous sources into a common data model. The biggest restriction to their application is that these tools and frameworks rely on the availability of meta-information in the form of ontologies, i.e., formal representations of concepts within a domain and the relationships between those concepts. While our framework can include semantic information, as shown in Section IV-A, it is not a fixed prerequisite.

In conclusion, we can say that our framework tries to fit into a middle ground between record linkage and semantic matching. We use methods applied in both areas but we leave the user the flexibility to choose, which of the features are actually needed in a mapping project.

III. Generic Modular Mapping framework

The Generic Modular Mapping Framework (GEMMA) is designed to be a flexible multi-purpose tool for any problem that requires matching data points to each other. The following subsections will introduce the requirements that were considered during the GEMMA development, the artifacts that make up the core idea behind GEMMA, describe the kind of mapping rules that can be implemented, show the generic process for the usage of GEMMA and describe the software architecture and the current GEMMA implementation.

A. Mapping

The basic challenge as defined in the introduction is the mapping of data that do not necessarily match completely in name, type, multiplicity or other details from different data sources to each other as depicted by Figure 1.

Relations between data from different sources and possibly in different formats need to be created. It must be possible to output the generated relations in a user-defined format. This leads to a first draft for a mapping tool, as depicted by Figure 2.

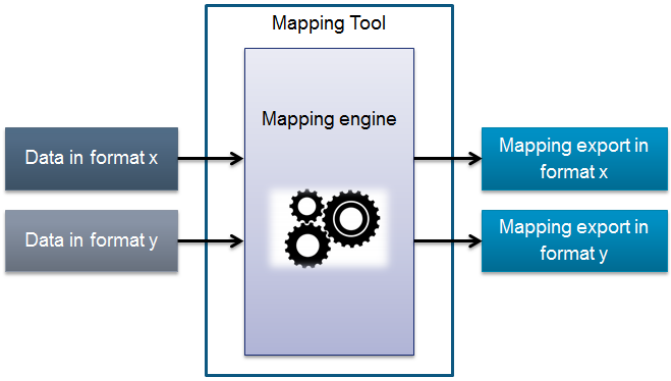

Figure 2. Mapping tool

The mapping tool shall be able to read data from different sources in different formats, then a mapping engine shall be able to create relations between the data and export these relations in different formats.

This is the minimum functionality that such a tool shall provide. In addition to that, there are three major requirements regarding the characteristics of the mapping tool, being **generic** in order to enable applications in different areas with similar challenges; being **modular**, as well as being **interactive**.

B. Generic

The requirement for a generic tool stems from the fact that different mapping problems and challenges require different data sources and mapping rules.

This means that the tool shall allow the user to define the rules that govern the creation of mappings. The tool will need to read and interpret such rules in order to be able to create mappings between different input data sets.

Additionally, it shall be possible to setup the current configuration of the mapping tool by means of user-defined configuration. Such a project configuration will contain information, such as where the input data is located, what mapping rules should be used and where the mapping export data should be written to.

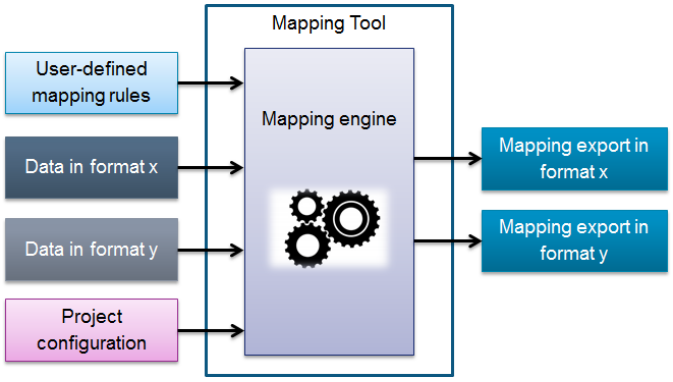

Figure 3. Generic mapping tool

The discussion above leads to an extension of the first draft of the mapping tool which is shown in Figure 3.

C. Modular

The requirement for modular software is an extension for the requirement that the software needs to be generic (see Section III-B). Modular programming is a software design technique that emphasizes separating the functionality of a program into independent, interchangeable modules, such that each contains necessary information for executing only one aspect of the desired functionality if required.

We anticipate using the mapping tool in very different contexts and applications with diverse data formats for import and export. To support this, the architecture needs to be modular.

Predefining the interfaces for importer and exporter modules allows creating new modules for specific applications without affecting the rest of the tool. Which modules are used in a specific mapping project can then be defined by the project configuration. A further benefit of the modular architecture is a separation of concerns and responsibilities. Different modules can be created and maintained by different developers or even organizations.

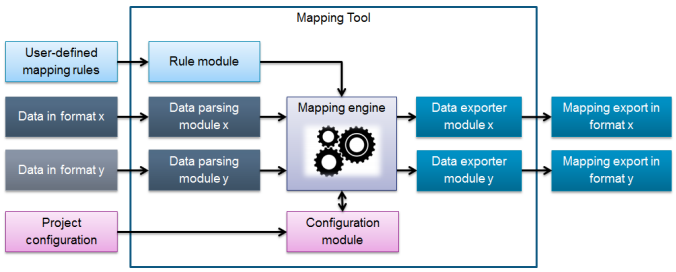

Figure 4. Generic modular mapping tool

The requirement for a modular tool affects the internal architecture of the mapping tool that is shown in Figure 4.

Furthermore, it should be possible to add, change or remove modules from the mapping tool without changes to the core application code. This allows for a packaging of the tool according to user and application needs and enables developing modules that must not be shared, e.g., for confidentiality reasons.

D. Interactive

Based on the assumption that the data in different data sources to be mapped can differ quite substantially in name, type, multiplicity or other details, it is reasonable to assume that a perfect mapping is not always possible. This directly leads to the requirement that the mapping tool needs to be interactive, i.e., allow user-involvement when needed.

An interactive tool displays information to the user and allows user to modify the displayed data. In our setting, this means that the mapping tool shall be able to display the generated mappings between the input data and allow the user to modify these mappings using a Graphical User Interface (GUI).

In order to present the generated mapping data to the user in a meaningful way it is necessary to consider interpretation of the generated mapping data. This requires an additional module: the resolver module. The resolver, as an application-specific module, is aware of application-specific requirements and features. Using this information, the resolver can process the generated mapping data and provide information regarding the application-specific validity of the generated mapping data to users.

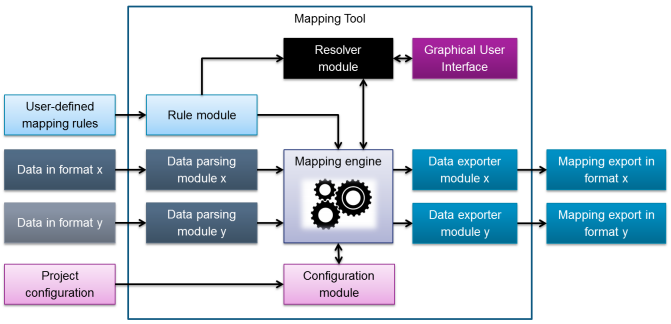

Figure 5. Interactive generic modular mapping tool

The requirement for an interactive tool leads to a change in the tool concept as shown in Figure 5. To further support the idea of a generic and flexible tool for different applications, the GUI module will be optional, i.e., it should be possible to run the mapping tool with or without the GUI.

E. Artefacts

GEMMA is centred around a set of core concepts that are depicted by Figure 6. In an effort to increase the flexibility of GEMMA, the core concepts have been defined in an abstract fashion.

The following artefacts are used:

- **Node** - Something that has properties that can be mapped to some other properties.

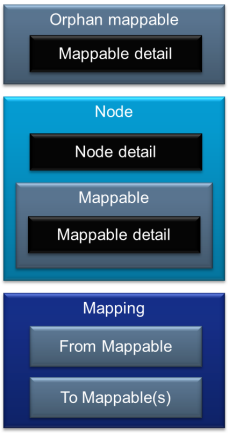

Figure 6. Overview of relevant artefacts

- **Mappable** - Something that can be mapped to some other thing according to specified mapping rules. Orphan mappables are mappables whose owning node is not known or not relevant to the problem.
- **Mapping** - The result of the application of mapping rules, i.e., a relation between one FROM mappable and one or more TO mappables. Note that the semantic interpretation of a mapping highly depends on the application scenario.
- **Mapping rule** - A function that specifies how mappings are created, i.e., how one mappable can be related to other mappables.
- **Mappable or node detail** - Additional attribute of a mappable or a node in the form of a {detail name:detail value} pair. Details are optional and can be defined in the context of a specific application scenario.

To illustrate these abstract definitions, Figure 7 provides a simple example, where real-world objects depicted on the left hand side are represented on the right hand side in the form of our GEMMA concepts.

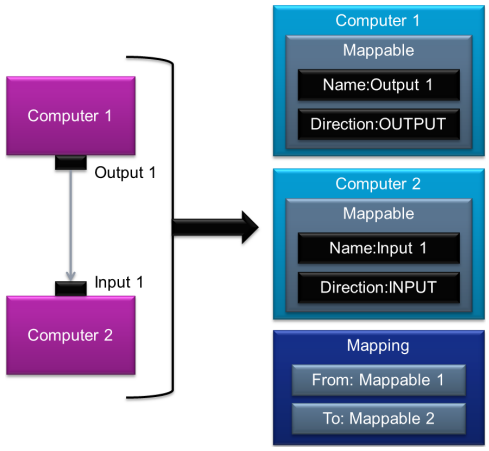

Figure 7. Simple example

In this context, the abstract concept definitions provided above are interpreted as follows:

- **Node** - A computer with input and output ports
- **Mappable** - An input or output port of a computer
- **Mapping** - The connection between ports
- **Mapping rule** - Output ports must be connected to input ports according to some specified criteria such as having the same port name or the same data type.
- **Mappable detail** - Every port has a detail called direction, which defines if the port is an input of output port of the computer

F. Mapping rules

One goal of GEMMA is to allow a large degree of freedom regarding the definition of the mapping rules, so that the framework can be used flexibly for very different kind of application scenarios. So far, the following kinds of mapping rules have been identified and are supported by GEMMA:

- **Exact matching**, e.g., map a mappable to other mappables with the exact same name.
- **Fuzzy matching**, or other forms of **approximate string matching** [9], e.g., map a mappable to other mappables with a similar name (similarity can be based on the Levenshtein distance (LD) [10], i.e., ”map” can be matched to ”mop” if we allow an LD of 1).
- **Wildcard matching**, e.g., map a mappable to mappables that contain a certain value.
- **RegEx matching**, e.g., map a mappable to mappables based on a regular expression.
- **Tokenized matching**, e.g., split a mappable property into tokens and then map to another mappable with a property that contains each of these tokens in any order.
- **Details**, e.g., map a mappable with value of detail X=x to other mappables with values of details Y=y and Z=z or more concretely, map a mappable with detail direction=”output” to mappables with detail direction=”input”.
- **Structured rewriting** of search term based on name, details and additional data, e.g., construct a new string based on the properties of a mappable and some given string parts and do a name matching with the new string (e.g., new string = ”ABCD::” + $mappable.detail(DIRECTION) + ”::TBD::” + $mappable.detail(LOCATION) would lead to a search for other mappables with the name ”ABCD::Input::TBD::Front”).
- **Semantic annotations** such as user-predefined potential mappings (bindings) using mediators as described in [11], e.g., map a mappable whose name is listed as a client of a mediator to all mappables whose name is listed as a provider of the same mediator.

And, of course, any combination of the above mentioned kinds of rules can be used. For example, structured rewriting could also be applied on the target mappables, which would in effect mean defining aliases for every mappable in the mappable database in the context of a rule.

In one GEMMA rule set, several rules can be defined for the same mappable with options for defining their application, e.g., only if the rule with the highest priority does not find any matches then rules with a lower priority are evaluated.

G. Process

The process for the usage of GEMMA is generic for all kinds of applications scenarios and consists of five steps:

1) Import
2) Pre-processing
3) Matching
4) Post-processing
5) Export

The mapping process is configured using an Extensible Markup Language (XML) configuration file that defines which parsers, rules, resolvers and exporters (see Section III-H for a detailed explanation of the terms) will be used in the mapping project. The open character of GEMMA allows implementing different data parsers for importing data, resolvers for post-processing of the mappings and data exporters for exporting data.

Import loads data into the framework. GEMMA provides the interfaces *DataParser*, *MappableSource* and *NodeSource* to anyone who has the need to define a new data parser for an application-specific configuration of GEMMA. All available parsers are registered in an internal parser registry where the Run Configuration can instantiate, configure and run those parsers, which are required by the configuration file. The data will then be stored in the mappable database. As our mappable database uses the full-text search engine Lucene [12], all relevant information from a mappable must be converted into Strings. Each mappable is assigned a unique identifier (ID) from its parser and other required information is stored as detail-value pair in so-called fields as shown in Figure 8.

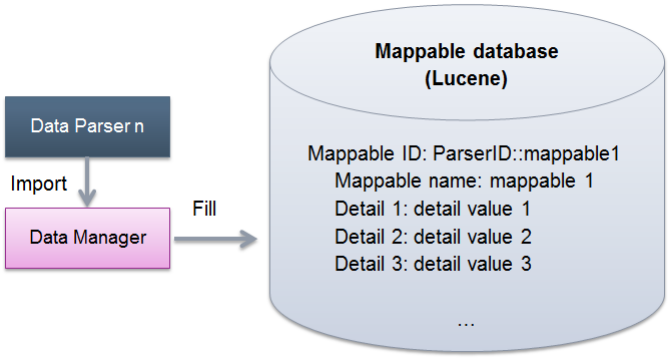

Figure 8. Import process

Pre-processing of data involves selection of mappables that will require matching using whitelists and/or blacklists and structured rewriting of, e.g., mappable names based on mappable details. Pre-processing will be user-defined in a set of rules in a file that can be edited with a standard text editor and does not require programming knowledge. The set of rules that should be applied in one mapping project will be defined by the configuration.

Matching involves running queries on the mappable database to find suitable matches for each mappable that is selected for mapping. The queries are derived from the mapping rules. A mapping is a one to (potentially) many

relation between one mappable and all the matches that were found.

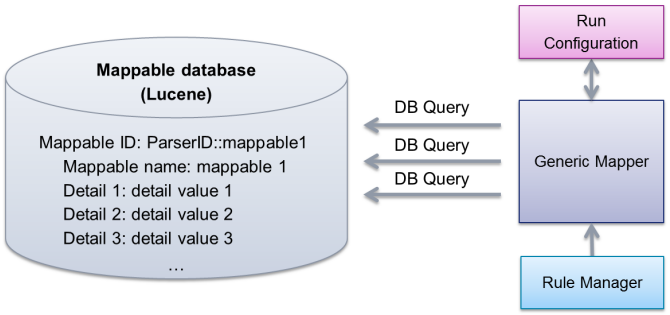

Figure 9. Matching process

As depicted by Figure 9, during the matching process, the generic mapper requests the list of existing mappables from the data manager. For every mappable the mapper retrieves applicable rules from the rule manager and generates queries that are run on the mappable database. The mapper then creates a mapping from the original mappable to the mappables yielded by the query result. The mappings are stored in the data manager.

Post-processing or match resolving is an optional step that is highly driven by the specific application as will be shown in Section IV. It potentially requires the interaction with the user to make a selection, e.g., a mapping rule might say that for a mappable only a one-to-one mapping is acceptable but if more than one match was found then the user must decide which should be selected. Post-processing also allows the user to apply the graphical user interface to review and validate the generated mapping results, to check the completeness and correctness of the defined rules, and to modify mappings manually, e.g., remove a mappable from a mapping if the match was not correct or create a new mapping manually.

Export is also highly application-specific. Exporting involves transformation of the internal data model into an application-specific output file. Similar to the *DataParser* interface, a generic *MappingExporter* interface allows the definition of custom exporters that are registered in an exporter registry where they can be accessed by the run configuration as dictated by the configuration file.

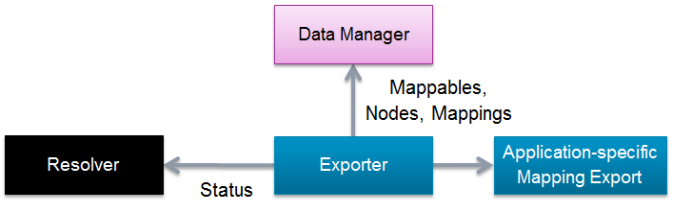

Figure 10. Export process

Each exporter can obtain the available mappables, nodes and mappings from the data manager and the resolver provides an exporter with the status of the elements as depicted by Figure 10. Using this information the exporter creates a mapping export. The mapping export can take many forms, e.g., it can be just an XML file as the standard exporter produces but it can also be an export directly into an application using the application's application programming interface (API). How the data is exported is completely encapsulated in the exporter.

H. Architecture and implementation

As already stated before, the Generic Mapping Framework is designed as a flexible answer to all sorts of mapping problems. This is represented in the architecture of the framework, which is depicted in Figure 11 in a simplified fashion. GEMMA modules can be categorized either as core or as application-specific. Core components are common for all GEMMA usage scenarios whereas the application specific components have to be developed to implement features that are very specific to achieve a certain goal. For example, data parsers are application-specific as applications might need data from different sources whereas the mappable database and query engine is a core component that is shared. Table I provides a brief description of the most important modules in GEMMA and their categorization.

TABLE I. GENERIC MAPPING FRAMEWORK MODULES

Module	Description	Core	Specific
Data Parser	Reads data (nodes and/or mappables) into the internal data model and feeds the mappable database		x
Mapper	Generates mappings between mappables based on rules	x	x
GUI	Interface for loading configuration, displaying mappings as well as allowing user-decisions and displaying of data based on resolver as shown in Figure 12	x	
Mappable Database	Stores mappable information and allows searches	x	
Data Manager	Stores mappables, nodes and mappings	x	
Resolver	Resolves mappings based on application specific semantics		x
Rule Manager	Reads mapping rules and provides rules information to other components	x	
Run Configuration	Holds the configuration that defines which parsers, exporters, mapper and rules are used in the current mapping project	x	
Data Exporter	Exports the internal data model into a specific file format		x

GEMMA is implemented in Java. As much as possible, open source libraries and frameworks have been used. The choice for the mappable database, for example, fell on Apache Lucene [12]. Lucene is is a high-performance, full-featured text search engine library. The choice of Lucene might seem odd because we are not using it for its originally intended purpose, indexing and searching of large text files, but it offers a lot of the search capabilities like fuzzy name matching that we need and is already in a very stable state with a strong record of industrial applications.

GEMMA was built on top of the Eclipse Rich Client Platform (RCP) [13], which is a collection of frameworks that enables building modular, pluggable architectures. As shown in Figure 13, the RCP provides some base services on top of which it is possible to build a custom application that may consist of a number of modules that work together in a flexible fashion.

GEMMA is an Eclipse product and uses the Eclipse Open Service Gateway Initiative (OSGi) extension mechanism [14] for registering and instantiating modules. This means that, as

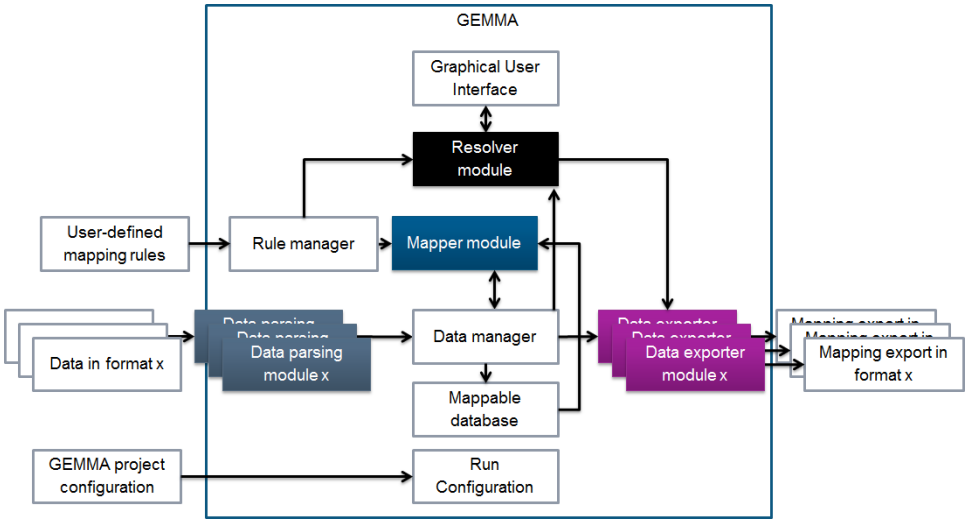

Figure 11. Generic mapping framework architecture

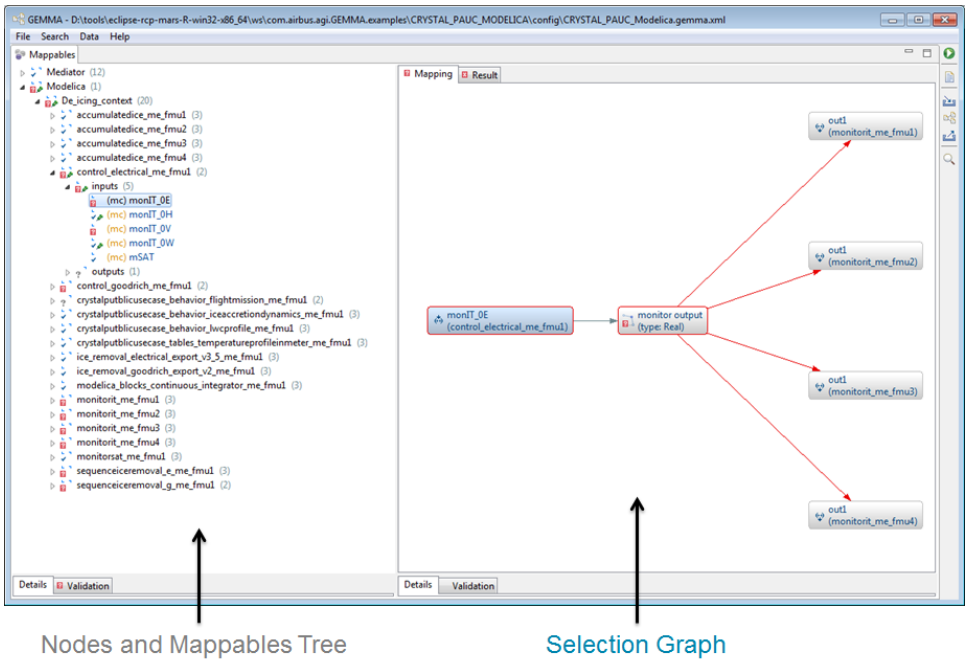

Figure 12. GEMMA graphical user interface

depicted by Figure 14, GEMMA is in essence a collection of Eclipse plugins, some of which can be selected by a user for specific applications, such as the data parsers or the exporters and some of which are fixed, such as the GUI. This architecture allows a tailored deployment of GEMMA.

If some modules are not needed by a user of if a module must not be given to some users, it is possible to remove the plugin from the installation directory of GEMMA without the need for any programming. Only the plugins that are required by a mapping project configuration are needed and instantiated during runtime as shown in Figure 15.

IV. Evaluation

The evaluation so far has been done using two application cases, simulation model composition and test bench setup.

In each of the application cases, four criteria have been evaluated to determine the success of the application of the mapping tool in the use case: mapping rates, adaptability, usability, performance.

- The **mapping rates criterion** includes the number of correct mappings, the number of incorrect mappings (false positives) and the number of missed mappings. As the difficulty of the mapping challenge depends on the characteristics of the input data it is not possible to define thresholds that determine a success of the

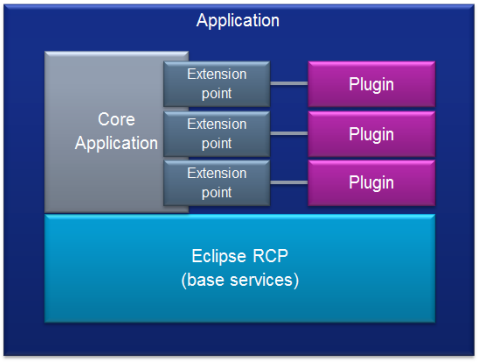

Figure 13. Eclipse RCP

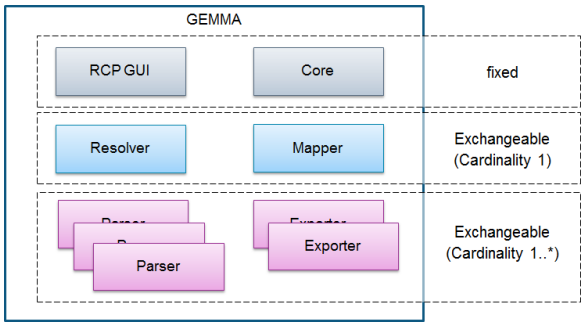

Figure 14. GEMMA as a collection of Eclipse plugins

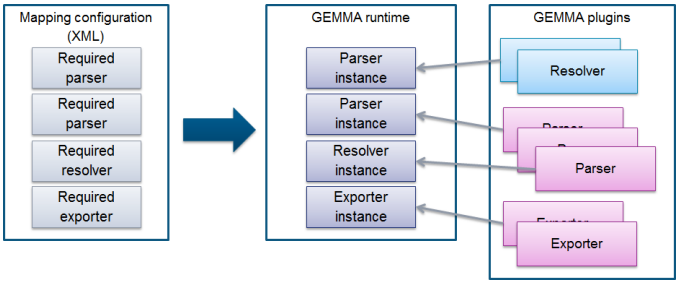

Figure 15. Instantiation of GEMMA modules at runtime

application.

- The **adaptability criterion** is not a measurable criterion. It is a subjective criterion to evaluate how easily and efficiently the mapping tool could be adapted to the needs of a new use case. This mainly focuses on the effort for the definition and validation of the mapping rules as well as the effort for creating or adapting modules that are required by the use case and their integration in the tool.
- Similar to the adaptability criterion, the **usability criterion** is based on feedback from the tool users and their subjective assessment of the effectiveness of using the tool.
- The **performance criterion** mainly refers to speed in terms of tool runtime: runtime for data parsing, mapping, resolving and exporting. As with the mapping rates criterion, a threshold for performance metrics cannot be defined a priori due to the diverse nature of the tool.

A. *Simulation model composition*

The description of the application case simulation model composition requires the introduction of the bindings concept as presented in [11]. The purpose of bindings is to capture the minimum set of information required to support model composition by an automated binding or connecting mechanism. For example, for the outputs of a given component, we wish to identify the appropriate inputs of another component to establish a connection.

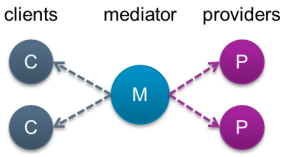

Figure 16. Bindings concept

To this end [11] introduces the notions of clients and providers. Clients require certain data; providers can provide the required data. However, clients and providers do not know each other a priori. Moreover, there may be multiple clients that require the same information. On the other hand, data from several providers may be needed in order to compute data required by one client. This results in a many-to-many relation between clients and providers. In order to associate the clients and the providers to each other the mediator concept is introduced, which is an entity that can relate a number of clients to a number of providers, as illustrated in Figure 16. References to clients and providers are stored in mediators in order to avoid the need for modifying client or provider models.

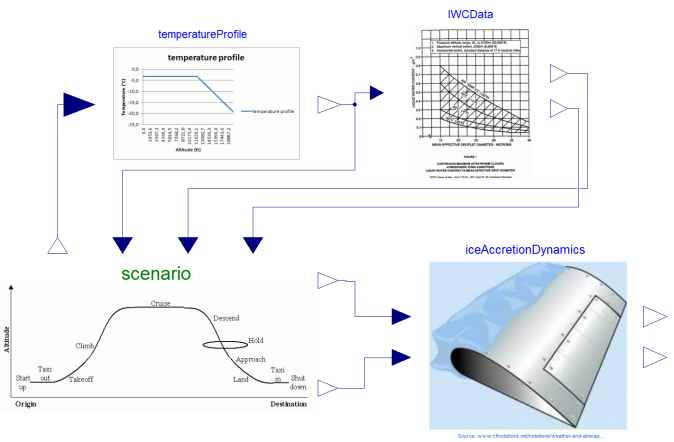

Figure 17. Assembled ice-accretion simulation based on [15]

After the bindings concept was introduced we can now turn to the description of the application case. Generally speaking, the application case is the automatic creation of connections between different model components in a model. Typically in modelling tools, to create a connection between one port of one component to another port of another component requires the user to draw each connection as one line from one port to the other port. If the components' interfaces or the model

structure change, then all of the connections have to be checked and some of them have to be redrawn. If we consider a large set of models that have to be changed frequently or if we want to create the models dynamically, then the effort for creating and maintaining the connections between the components in the models becomes a serious issue. The goal of our application case is the formalization of the often implicit rules which the user applies to create the connections to automate this process.

Consider the model depicted by Figure 17, which is a part of the model from the public aerospace use case in the CRYSTAL project [15]. It consists of component models such as flight scenario profile, ice accretion dynamics, and tables for temperature or liquid water content. All of the component models must be interconnected. For example, the temperature profile component requires the current aircraft altitude, which is provided by the flight scenario component; the ice accretions dynamics component requires the current aircraft speed, which is also provided by the scenario component, etc. The individual models were built using the Modelica tool Dymola and exported as Functional Mockup Units [16] (FMUs) in order to be integrated, i.e., instantiated and connected, in a co-simulation environment.

However, assume that the models were created without this specific context in mind. They neither have agreed interfaces, nor do the name and type of the component elements to be connected necessarily match. In order to be able to find the counter parts, i.e., to know that the input of the ice accretion instance should be connected to the appropriate output of the scenario model instance, a dedicated XML file captures some additional information. This way we can capture such interrelations without modifying the models. This data is used as follows: whenever the model ”IceAccretionDynamics” is instantiated, bind its input port ”aspeed” to the output ”port p_v”, which belongs to the instance of type ”ScenarioMissionProfile1”.

Whenever there will be another model that requires the same data, i.e., current aircraft speed, an additional client entry is added to the same mediator. Similarly, whenever there is another model that outputs this data, its corresponding element is referenced in a new provider entry. This approach in particular pays off as soon as there are several models that require or provide the same data. Their connection is then resolved whenever they are instantiated in a specific context model such as the one depicted in Figure 17.

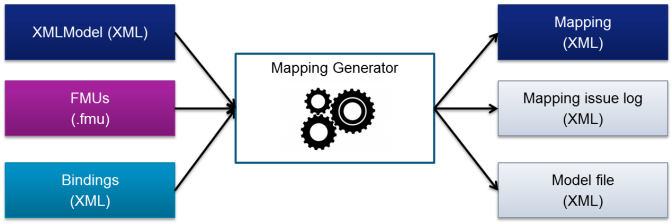

Figure 18. Mapping generator for simulation model composition

In our setting, the bindings specification XML file and the model XML file are application specific sources that are inputs to our generic mapping framework as depicted by Figure 18. The information read from these sources by the application specific parsers is put into the core module mappable database. Two rules are used to query the mappable database to find suitable matches for each mappable. The matching results are then given to the resolver module. This module is aware of the bindings concept and is able to resolve chains of matches and generate a binding for each client and, if necessary, involve the user when an unambiguous mapping is not possible automatically.

In the end, the mapping framework uses a list of FMUs, a description of the simulation model consisting of instances of classes implemented in the FMUs and a description of the bindings in the form of an XML file. The output is then the complete simulation model with all the connections between the simulation instances as sketched by Figure 19.

The evaluation of GEMMA in the simulation model composition application case was considered successful regarding all four evaluation criteria.

B. Test bench setup

The test bench setup application case is driven by the needs of test engineers. They are given a hardware System under Test (SuT), a formal definition of the interfaces of the SuT and other equipment and a description of the specified logic of the SuT, which should be tested. Unfortunately, the formal interface definition has been finalized after the specification of the logic, which means that the signal names in the logic description and the signal names in the formal interface definition, which has been implemented in the SuT, do not match. Today, a significant amount of manual effort is required to discover the correct formal signal name for every logical signal. To ease this, GEMMA has been configured as shown in Figure 20.

The goal of the application case is to find a mapping between the name of a signal used in the description of the SuT logic and the corresponding formal interface signal name as shown in Figure 21.

Since the names of the signals could be quite different, the test bench setup application case required the use of the structured rewriting rule type (see Section III-F). One of the rules for the test bench setup is depicted by Figure 22 in pseudo code. The rule defines a new local variable called *soughtName* whose content depends on some attributes of the mappable (enclosed in $$) and instead of searching for other mappables that have the same or a similar name as the original mappable, GEMMA searches now for mappables whose name is equal to the variable *soughtName*. If a mappable has the attributes *direction*, *type*, *BLOCKID* and *ID* with the respective values OUTPUT, SuT_Type1, 45 and 67 then *soughtName* would take the value *AB_BLOCK45_STATUS_67* and GEMMA will search for and map to another mappable in the database with that name.

The main challenge for this application case was the amount of data. Even for a small SuT, the mappable database contained 350000 mappables and matches had to be found for 2500 mappables. Nevertheless, the application proved to be successful. The total run time is around 30 seconds including the time for data import and export, and the average time per query is 4.5 ms on a standard PC.

The evaluation of GEMMA in the test bench setup application case was considered successful regarding all four evaluation criteria.

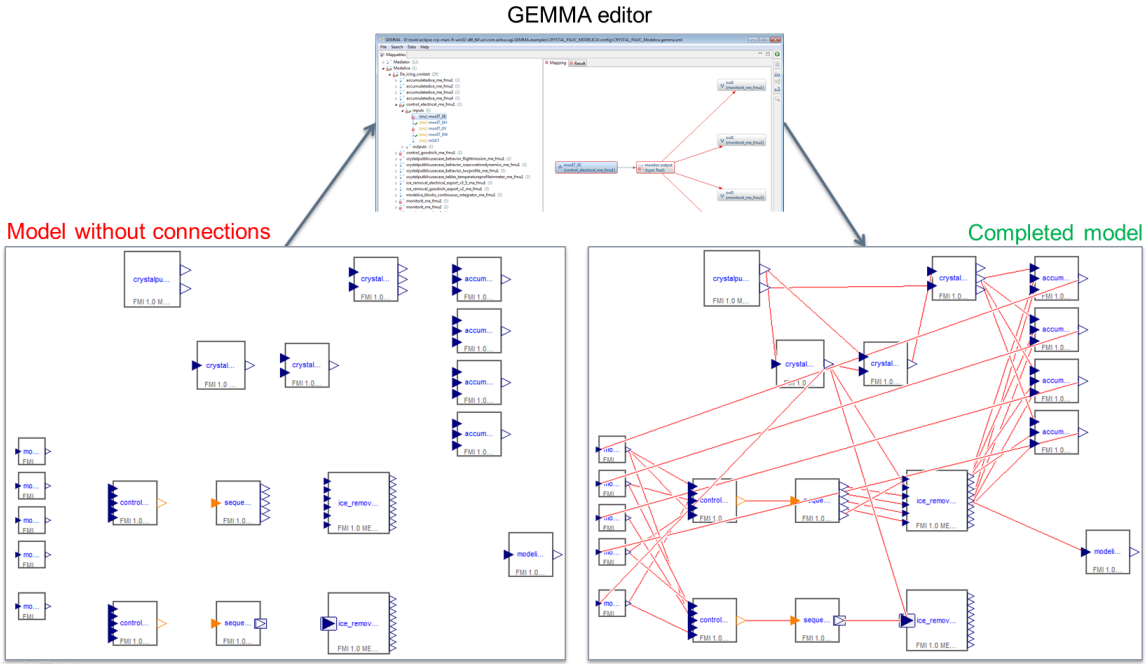

Figure 19. Input and output artefacts of simulation model composition mapper)

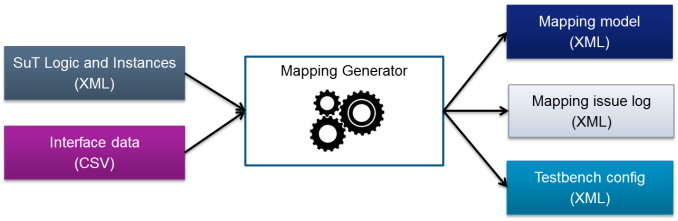

Figure 20. Mapping generator for test bench setup

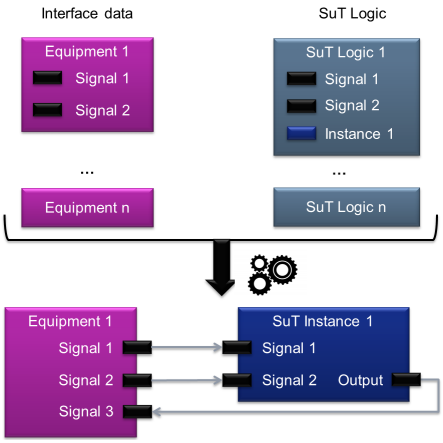

Figure 21. Input and output artifacts of test bench setup mapper

V. Conclusion

In this paper, we introduce a new framework for generic mapping problems, GEMMA. It is geared towards high flexibility for dealing with a number of very different challenges. To this end it has an open architecture that allows the inclusion of application-specific code for reading and exporting data and the resolving of mapping results. Furthermore, it provides a generic rule-based mapping engine that allows users without programming knowledge to define their own mapping rules. So far, the evaluation in the two application cases described in this paper has been highly successful.

The modular architecture based on the Eclipse RCP proved to be especially useful to allow using GEMMA for different purposes. The effort for adapting GEMMA for new applications, i.e., mainly the development of custom parser, resolver and exporter modules is low compared and usually takes a couple of days for an experienced Java developer. This is quite low compared to the effort required for the development of a new application that could satisfy the user needs.

As said in Section II, as far as we know, there is currently no other tool with the same functionality as GEMMA. This prevents a direct comparison in terms of performance of GEMMA with other solutions. For our future work we also plan to compare GEMMA functionally to other solutions that rely on more formalized semantic information in the form of ontologies. Depending on the results of this comparison, this might lead to an extension of GEMMA, so that in addition to the matching based on the Lucene text database there will be the possibility to include the results from a semantic reasoner in the matching process. Another possible extension of GEMMA that we are currently investigating is the inclusion of machine learning technology into GEMMA. Machine Learning could potentially be used to learn from existing mappings and to create or propose to the user new mappings based on that knowledge.

At the same time, we are actively looking for further application cases to mature the framework.

```
foreach SuTLogicInstance:instance
 foreach Mappable:mappable
  if mappable.detail(DIRECTION) == INPUT
   Search Mappable:otherMappables
     with mappable.detail(NAME) == otherMappable.detail(NAME)
   Create mapping from mappable to otherMappables
   elseif mappable.detail(direction) == OUTPUT
    if instance.detail(type) == SuT_Type1
     soughtName = "AB_BLOCK$$mappable.detail(BLOCKID)$$_STATUS_$$mappable.detail(ID)$$"
    elseif instance.detail(type) == SuT_Type2
     soughtName = "BLOCK$$mappable.detail(BLOCKID)$$_STATUS_$$mappable.detail(ID)$$_CD"
    endif
    Search Mappable:otherMappables
     with mappable.detail(NAME) == soughtName
    Create mapping from mappable to otherMappables
  endif
 endfor
endfor
```

Figure 22. One implemented rule for test bench setup (in pseudo code)

REFERENCES

[1] P. Helle and W. Schamai, “Using a generic modular mapping framework for simulation model composition,” in SIMUL 2015, The Seventh International Conference on Advances in System Simulation, 2015, pp. 72–78.

[2] H. L. Dunn, “Record linkage*,” American Journal of Public Health and the Nations Health, vol. 36, no. 12, 1946, pp. 1412–1416.

[3] I. P. Fellegi and A. B. Sunter, “A theory for record linkage,” Journal of the American Statistical Association, vol. 64, no. 328, 1969, pp. 1183–1210.

[4] Q. He, Z. Li, and X. Zhang, “Data deduplication techniques,” in Future Information Technology and Management Engineering (FITME), 2010 International Conference on, vol. 1. IEEE, 2010, pp. 430–433.

[5] H. Koepcke and E. Rahm, “Frameworks for entity matching: A comparison,” Data & Knowledge Engineering, vol. 69, no. 2, 2010, pp. 197 – 210.

[6] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, “Duplicate record detection: A survey,” Knowledge and Data Engineering, IEEE Transactions on, vol. 19, no. 1, 2007, pp. 1–16.

[7] F. Giunchiglia, A. Autayeu, and J. Pane, “S-match: An open source framework for matching lightweight ontologies,” Semantic Web, vol. 3, no. 3, 2012, pp. 307–317.

[8] Y. Hooi, M. Hassan, and A. Shariff, “A survey on ontology mapping techniques,” in Advances in Computer Science and its Applications, ser. Lecture Notes in Electrical Engineering, H. Y. Jeong, M. S. Obaidat, N. Y. Yen, and J. J. J. H. Park, Eds. Springer Berlin Heidelberg, 2014, vol. 279, pp. 829–836.

[9] P. A. Hall and G. R. Dowling, “Approximate string matching,” ACM computing surveys (CSUR), vol. 12, no. 4, 1980, pp. 381–402.

[10] V. I. Levenshtein, “Binary codes capable of correcting deletions, insertions, and reversals,” in Soviet physics doklady, vol. 10, no. 8, 1966, pp. 707–710.

[11] W. Schamai, P. Fritzson, C. J. Paredis, and P. Helle, “ModelicaML value bindings for automated model composition,” in Proceedings of the 2012 Symposium on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium. Society for Computer Simulation International, 2012, p. 31.

[12] M. McCandless, E. Hatcher, and O. Gospodnetic, Lucene in Action. Manning Publications Co., 2010.

[13] J. McAffer, J.-M. Lemieux, and C. Aniszczyk, Eclipse Rich Client Platform. Addison-Wesley Professional, 2010.

[14] R. Hall, K. Pauls, S. McCulloch, and D. Savage, OSGi in action: Creating modular applications in Java. Manning Publications Co., 2011.

[15] A. Mitschke et al., “CRYSTAL public aerospace use case Development Report - V2,” ARTEMIS EU CRYSTAL project, Tech. Rep. D208.902, 2015.

[16] T. Blochwitz et al., “The functional mockup interface for tool independent exchange of simulation models,” in 8th International Modelica Conference, Dresden, 2011, pp. 20–22.